Weighted Model Counting with Conditional Weights

Paulius Dilkas

1st August 2020

1 Introduction

- The main narrative
  1. Putting weights on literals is restrictive.
  2. To overcome this, it is common practice to add more literals.
  3. This is indeed always possible but much slower.
  4. We show how we can define conditional weights on literals, allowing us to encode any measure.
  5. We demonstrate a specific example of this by presenting a new way to encode Bayesian networks into instances of WMC and adapting a WMC algorithm (ADDMC) to run on the new format.
  6. We show that this results in significantly faster inference.
  7. We show that our encoding results in asymptotically fewer literals and fewer ADDs.
- Introduce
  - WMC [18]
  - ADDMC [10] and ADDs [1]
  - A Bayesian network is a directed acyclic graph with random variables as vertices that defines a probability distribution over them. The full probability distribution can be expressed as the product of the conditional probability distributions of all random variables. For discrete Bayesian networks (and we only consider discrete networks in this paper), each such distribution can be represented by a table known as a conditional probability table (CPT).

F What are the main claims, what are the main takeaways, intuitive [???] of theorems to follow.

- Our work is, in spirit, similar to... using ROBDDs for Inference in Bayesian Networks with Troubleshooting as an Example [16]: this is an interesting approach where deterministic parts of a BN are expressed as Boolean functions, i.e., extracting the logical from the probabilistic. Maybe I can make a case that my encoding (the textual version) kind of does the same, even though the algorithm doesn't.

2 Related Work

Using WMC to perform inference on Bayesian networks. Hitherto, four techniques have been proposed for encoding Bayesian networks into instances of WMC. We will identify them based on the initials of authors as well as publications years: `d02` [9], `sbk05` [18], `cd05` [3], and `cd06` [4]. Below we summarise the observed performance differences among them. Sang et al. [18] claim that `sbk05` is a smaller encoding than `d02` with respect to both the number of clauses and the number of variables but provide no experimental comparison. Chavira and Darwiche [3] compare `cd05` with `d02` by measuring the time it takes to compile either encoding into an arithmetic circuit (but do not measure inference time). The more recent encoding

Table 1: A comparison of Boolean-algebraic (BA) and set-theoretic (ST) concepts for 2^U for any set U

BA name	BA symbol	ST symbol	ST name
bottom	$\bot$	$\emptyset$	empty set
top	$\top$	U	
meet, and	$\wedge$	$\cap$	intersection
join, or	$\vee$	$\cup$	union
complement, not	$\neg$	^{c}	complement
	$\leq$	$\subseteq$	subset relation, set inclusion
atom			singleton, unit set

Table 2: Notation for a logic with two atoms. The elements in both columns are listed in the same order.

Name in logic	Boolean-algebraic notation	Set-theoretic notation
Atoms (elements of U)	a, b	a, b
Models (elements of 2^U)	$\neg a \wedge \neg b, a \wedge \neg b, \neg a \wedge b, a \wedge b$	$\emptyset, \{a\}, \{b\}, \{a, b\}$
Formulas (elements of 2^{2^U})	$\top$	$\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
	$\neg a \vee \neg b, a \rightarrow b$	$\{\emptyset, \{a\}, \{b\}\}, \{\emptyset, \{a\}, \{a, b\}\}$
	$b \rightarrow a, a \vee b$	$\{\emptyset, \{b\}, \{a, b\}\}, \{\{a\}, \{b\}, \{a, b\}\}$
	$\neg b, \neg a, a \leftrightarrow b$	$\{\emptyset, \{a\}\}, \{\emptyset, \{b\}\}, \{\emptyset, \{a, b\}\}$
	$(a \wedge \neg b) \vee (b \wedge \neg a), a, b$	$\{\{a\}, \{b\}\}, \{\{a\}, \{a, b\}\}, \{\{b\}, \{a, b\}\}$
	$\neg a \wedge \neg b, a \wedge \neg b, \neg a \wedge b, a \wedge b$	$\{\emptyset\}, \{\{a\}\}, \{\{b\}\}, \{\{a, b\}\}$
	$\bot$	$\emptyset$

`cd05` always compiles faster and results in a smaller arithmetic circuit (as measured by the number of edges). In their subsequent paper, the same authors perform two sets of experiments (that are relevant to this summary) [4]. First, they compile `cd05` and `cd06` encodings into d-DNNF (i.e., deterministic decomposable negation normal form [8]), measuring both compilation time and numbers of edges in the d-DNNF diagram. The results are mostly in favour of `cd06`. Second, they compare the inference time of `sbk05` run with Cachet [17] with the compile times of `cd05` and `cd06`, but only on five (types of) instances. In these experiments, `cd06` is always faster than `cd05`, while the comparison with `sbk05` is mixed. Sometimes `cd06` is orders of magnitude faster than `sbk05`, sometimes slightly slower. The performance difference between `sbk05` and `cd05` is even harder to judge: `sbk05` is better on three out of five instances and worse on the remaining two. Based on this description, one would expect `cd06` to be faster than both `cd05` and `sbk05`, both of which should be faster than `d02`. The experiments in Section 6, however, strongly disagree with this prediction, showing that the quality of an encoding depends strongly on the underlying search algorithm or compilation technique.

ADDs and their use in probabilistic inference. ADDs provide an efficient way to manipulate functions from a Boolean algebra to any algebraic structure (most commonly the real numbers) [1]. They have been used to represent value functions in Markov decision processes [13] and, for Bayesian network inference, to represent each CPT as an ADD [21] as well as by combining ADDs and arithmetic circuits into a single representation [5]. ADDs are particularly advantageous in this situation because of their ability to fully exploit context-specific independence, i.e., observable structure within a CPT that is not inherited from the structure of the Bayesian network [2].

3 Boolean Algebras, Power Sets, and Propositional Logic

Let $\mathcal{L}$ be a propositional logic with atoms a and b, and let $U = \{a, b\}$. Then 2^U, the power set of U, is the set of all models of $\mathcal{L}$, and 2^{2^U} is the set of all formulas. These sets can also be represented as Boolean algebras—see Table 1 for an overview of notational differences and Table 2 for examples of how various elements can be represented in both notations. Most importantly, note that the word *atom* has completely different meanings in logic and in Boolean algebras. An atom in $\mathcal{L}$ is an atomic formula, i.e., an element of U, whereas an atom in a Boolean algebra is (in set-theoretic terms) a singleton set. For instance, an atom in 2^{2^U} corresponds to a model of $\mathcal{L}$, i.e., an element of 2^U. Unless referring specifically to a logic, we will use the algebraic definition of an atom while referring to logical atoms as *variables*. In the rest of the paper, for any set U, we will use set-theoretic notation for 2^U and Boolean-algebraic notation for 2^{2^U}, except for (Boolean) atoms in 2^{2^U} that are denoted as $\{x\}$ for some model $x \in 2^U$.

3.1 The Space of Functions on Boolean Algebras

We build on the definitions of multiplication and projection in the ADDMC paper [10] and define more operations that can be used to manipulate functions from Boolean algebras to non-negative real numbers. All of these operations have efficient implementations in the CUDD [20] package for manipulating ADDs (among other things) that is used by ADDMC [10].

Definition 1 (Operations on functions)**.** Let $\alpha\colon 2^X \to \mathbb{R}_{\geq 0}$ and $\beta\colon 2^Y \to \mathbb{R}_{\geq 0}$ be functions, $p \in \mathbb{R}_{\geq 0}$, and $x \in X$. We define the following operations:

Addition: $\alpha + \beta\colon 2^{X \cup Y} \to \mathbb{R}_{\geq 0}$ is such that $(\alpha + \beta)(T) = \alpha(T \cap X) + \beta(T \cap Y)$ for all $T \in 2^{X \cup Y}$.

Multiplication: $\alpha \cdot \beta\colon 2^{X \cup Y} \to \mathbb{R}_{\geq 0}$ is such that $(\alpha \cdot \beta)(T) = \alpha(T \cap X) \cdot \beta(T \cap Y)$ for all $T \in 2^{X \cup Y}$.

Scalar multiplication: $p\alpha\colon 2^X \to \mathbb{R}_{\geq 0}$ is such that $(p\alpha)(T) = p \cdot \alpha(T)$ for all $T \in 2^X$.

Complement: $\overline{\alpha}\colon 2^X \to \mathbb{R}_{\geq 0}$ is such that $\overline{\alpha}(T) = 1 - \alpha(T)$ for all $T \in 2^X$.

Projection: $\exists_x \alpha\colon 2^{X \setminus \{x\}} \to \mathbb{R}_{\geq 0}$ is such that $(\exists_x \alpha)(T) = \alpha(T) + \alpha(T \cup \{x\})$ for all $T \in 2^{X \setminus \{x\}}$.

Observation 1. *Let U be a set, and $\mathcal{V} = \{\alpha\colon 2^X \to \mathbb{R}_{\geq 0} \mid X \subseteq U\}$. Then $\mathcal{V}$ is a semi-vector space with three additional operations: (non-scalar) multiplication, complement, and projection. Specifically, note that both addition and multiplication are both associative and commutative.*

Definition 2 (Special functions)**.** We define several special functions:

- unit $1\colon 2^\emptyset \to \mathbb{R}_{\geq 0}$, $1(\emptyset) = 1$;
- zero $0\colon 2^\emptyset \to \mathbb{R}_{\geq 0}$, $0(\emptyset) = 0$;
- and $[a]\colon 2^{\{a\}} \to \mathbb{R}_{\geq 0}$, $[a](\emptyset) = 0$, $[a](\{a\}) = 1$ for any a.

Henceforth, for any function $\alpha\colon 2^X \to \mathbb{R}_{\geq 0}$ and any set T, we will write $\alpha(T)$ to mean $\alpha(T \cap X)$.

4 WMC as a Measure

Let U be a set. A *measure* is a function $\mu\colon 2^{2^U} \to \mathbb{R}_{\geq 0}$ such that $\mu(\bot) = 0$, and $\mu(a \vee b) = \mu(a) + \mu(b)$ for all $a, b \in 2^{2^U}$ whenever $a \wedge b = \bot$. A *weight function* is a function $\nu\colon 2^U \to \mathbb{R}_{\geq 0}$. A weight function is *factored* if $\nu = \prod_{x \in U} \nu_x$ for some functions $\nu_x\colon 2^{\{x\}} \to \mathbb{R}_{\geq 0}$, $x \in U$. We say that a weight function $\nu\colon 2^U \to \mathbb{R}_{\geq 0}$ *induces* a measure $\mu_\nu\colon 2^{2^U} \to \mathbb{R}_{\geq 0}$ if

$$\mu_\nu(x) = \sum_{\{u\} \leq x} \nu(u). \tag{1}$$

Finally, a measure $\mu\colon 2^{2^U} \to \mathbb{R}_{\geq 0}$ is *factorable* if there exists a factored weight function $\nu\colon 2^U \to \mathbb{R}_{\geq 0}$ that induces μ.

Lemma 1. *The function μ_ν, as defined by Eq. (1), is a measure.*

Proof. Note that $\mu_\nu(\bot) = 0$ since there are no atoms below $\bot$. Let $a, b \in 2^{2^U}$ be such that $a \wedge b = \bot$. By elementary properties of Boolean algebras, all atoms below $a \vee b$ are either below a or below b. Moreover, none of them can be below both a and b because then they would have to be below $a \wedge b = \bot$. Thus

$$\mu_\nu(a \vee b) = \sum_{\{u\} \leq a \vee b} \nu(u) = \sum_{\{u\} \leq a} \nu(u) + \sum_{\{u\} \leq b} \nu(u) = \mu_\nu(a) + \mu_\nu(b)$$

as required. □

In this formulation, the process of calculating the value of $\mu_\nu(x)$ for some $x \in 2^{2^U}$ with a given definition of ν is known as WMC.

Relation to the classical (logic-based) view of WMC. Let $\mathcal{L}$ be a propositional logic with two atoms a and b as in Section 3 and $w\colon \{a, b, \neg a, \neg b\} \to \mathbb{R}_{\geq 0}$ a *weight function* defined as $w(a) = 0.3$, $w(\neg a) = 0.7$, $w(b) = 0.2$, $w(\neg b) = 0.8$. Furthermore, let Δ be a theory in $\mathcal{L}$ with a sole axiom a. Then Δ has two models: $\{a, b\}$ and $\{a, \neg b\}$ and its WMC [6] is

$$\mathrm{WMC}(\Delta) = \sum_{\omega \models \Delta} \prod_{\omega \models l} w(l) = w(a)w(b) + w(a)w(\neg b) = 0.3. \tag{2}$$

Alternatively, we can define $\nu_a\colon 2^{\{a\}} \to \mathbb{R}_{\geq 0}$ as $\nu_a(\{a\}) = 0.3$, $\nu_a(\emptyset) = 0.7$ and $\nu_b\colon 2^{\{b\}} \to \mathbb{R}_{\geq 0}$ as $\nu_b(\{b\}) = 0.2$, $\nu_b(\emptyset) = 0.8$. Let μ be the measure on 2^{2^U} induced by $\nu = \nu_a \cdot \nu_b$. Then, equivalently to Eq. (2), we can write

$$\mu(a) = \nu(\{a, b\}) + \nu(\{a\}) = \nu_a(\{a\})\nu_b(\{b\}) + \nu_a(\{a\})\nu_b(\emptyset) = 0.3.$$

4.1 Not All Measures Are Factorable

Example 1. Let $U = \{a, b\}$ be a set of atoms and $\mu\colon 2^{2^U} \to \mathbb{R}_{\geq 0}$ a measure defined as:[1]

$$\begin{aligned} \mu(a \wedge b) &= 0.72, \\ \mu(a \wedge \neg b) &= 0.18, \\ \mu(\neg a \wedge b) &= 0.07, \\ \mu(\neg a \wedge \neg b) &= 0.03. \end{aligned}$$

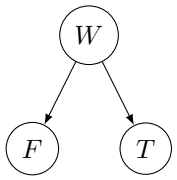
If μ could be represented using traditional (factored) WMC, we would have to find two weight functions $\nu_a\colon 2^{\{a\}} \to \mathbb{R}_{\geq 0}$ and $\nu_b\colon 2^{\{b\}} \to \mathbb{R}_{\geq 0}$ such that $\nu = \nu_a \cdot \nu_b$ induces μ, i.e., ν_a and ν_b would have to satisfy this system of equations:

$$\begin{aligned} \nu_a(\{a\}) \cdot \nu_b(\{b\}) &= 0.72 \\ \nu_a(\{a\}) \cdot \nu_b(\emptyset) &= 0.18 \\ \nu_a(\emptyset) \cdot \nu_b(\{b\}) &= 0.07 \\ \nu_a(\emptyset) \cdot \nu_b(\emptyset) &= 0.03, \end{aligned}$$

which has no solutions.

Alternatively, we can let b depend on a and consider weight functions $\nu_a\colon 2^{\{a\}} \to \mathbb{R}_{\geq 0}$ and $\nu_b\colon 2^{\{a,b\}} \to \mathbb{R}_{\geq 0}$ defined as $\nu_a(\{a\}) = 0.9$, $\nu_a(\emptyset) = 0.1$, and $\nu_b(\{a, b\}) = 0.8$, $\nu_b(\{a\}) = 0.2$, $\nu_b(\{b\}) = 0.7$, $\nu_b(\emptyset) = 0.3$. One can easily check that with these definitions ν indeed induces μ.

[1] The value of μ on any other element of the Boolean algebra can be deduced using the definition.



w	$\Pr(W = w)$
1	0.5
0	0.5

w	f	$\Pr(F = f \mid W = w)$
1	1	0.6
1	0	0.4
0	1	0.1
0	0	0.9

w	t	$\Pr(T = t \mid W = w)$
1	l	0.2
1	m	0.4
1	h	0.4
0	l	0.6
0	m	0.3
0	h	0.1

Figure 1: An example Bayesian network with its CPTs

Note that in this case we chose to interpret ν_b as $\Pr(b \mid a)$ while—with a different definition of ν_b that represents the joint probability distribution $\Pr(a, b)$—ν_b by itself could induce μ. In general, however, factorising the full weight function into several smaller functions often results in weight functions with smaller domains which leads to increased efficiency and decreased memory usage [10].

Add the proof that one can always go from any measure to a factorable measure by adding more literals and say: Not all measures are factorable but it is common practice to add more literals to make the measure factorable. We show that it's always possible and yet it can be faster not to do it.

5 Encoding Bayesian Networks Using Conditional Weights

In this section, we describe a way to encode Bayesian networks into WMC without restricting oneself to factorable measures and thus having to add extra variables to maintain literal independence. In line with the names of previous encodings, we shall call this encoding `db20`.

Recall that a Bayesian network is a directed acyclic graph with random variables as vertices. Let $\mathcal{V}$ denote the set of random variables. For any random variable $X \in \mathcal{V}$, let $\operatorname{im} X$ denote its set of possible values and $\operatorname{pa}(X)$ the set of its parents. The full probability distribution is then equal to $\prod_{X \in \mathcal{V}} \Pr(X \mid \operatorname{pa}(X))$. See Fig. 1 for an example Bayesian network which we will refer to throughout this section. For this network, $\mathcal{V} = \{W, F, T\}$, $\operatorname{pa}(W) = \emptyset$, $\operatorname{pa}(F) = \operatorname{pa}(T) = \{W\}$, $\operatorname{im} W = \operatorname{im} F = \{0, 1\}$, and $\operatorname{im} T = \{l, m, h\}$.

Definition 3 (Indicator variables)**.** Let $X \in \mathcal{V}$ be a random variable. If X is binary (i.e., $|\operatorname{im} X| = 2$), we can arbitrary identify one of the values as 1 and the other one as 0 (i.e, $\operatorname{im} X \cong \{0, 1\}$). Then X can be represented by a single *indicator variable* $\lambda_{X=1}$. For notational simplicity, for any set S, we write $\lambda_{X=0} \in S$ or $S = \{\lambda_{X=0}, \dots\}$ to mean $\lambda_{X=1} \notin S$.

On the other hand, if X is not binary, we represent X with $|\operatorname{im} X|$ indicator variables, one for each value. We let

$$\mathcal{E}(X) = \begin{cases} \{\lambda_{X=1}\} & \text{if } |\operatorname{im} X| = 2 \\ \{\lambda_{X=x} \mid x \in \operatorname{im} X\} & \text{otherwise.} \end{cases}$$

denote the set of indicator variables for X and $\mathcal{E}^*(X) = \mathcal{E}(X) \cup \bigcup_{Y \in \operatorname{pa}(X)} \mathcal{E}(Y)$ denote the set of indicator variables for X and its parents in the Bayesian network. Finally, let $U = \bigcup_{X \in \mathcal{V}} \mathcal{E}(X)$ denote the set of all indicator variables for all random variables in the Bayesian network.

For the Bayesian network in Fig. 1, this gives us:

$$\begin{aligned} \mathcal{E}(W) &= \{\lambda_{W=1}\}, \\ \mathcal{E}(F) &= \{\lambda_{F=1}\}, \\ \mathcal{E}(T) &= \{\lambda_{T=l}, \lambda_{T=m}, \lambda_{T=h}\}, \\ \mathcal{E}^*(W) &= \{\lambda_{W=1}\}, \\ \mathcal{E}^*(F) &= \{\lambda_{F=1}, \lambda_{W=1}\}, \\ \mathcal{E}^*(T) &= \{\lambda_{T=l}, \lambda_{T=m}, \lambda_{T=h}, \lambda_{W=1}\}. \end{aligned}$$

Algorithm 1: Encoding a Bayesian network as a function $2^U \to \mathbb{R}_{\geq 0}$

Data: a Bayesian network with vertices $\mathcal{V}$ and probability distribution Pr
Result: a function $\phi\colon 2^U \to \mathbb{R}_{\geq 0}$
$\phi \leftarrow 1$;
for $X \in \mathcal{V}$ **do**
  let $\operatorname{pa}(X) = \{Y_1, \ldots, Y_n\}$;
  $\mathrm{CPT}_X \leftarrow 0$;
  if $|\operatorname{im} X| = 2$ **then**
    for $(y_1, \ldots, y_n) \in \prod_{i=1}^n \operatorname{im} Y_i$ **do**
      $p_1 \leftarrow \Pr(X = 1 \mid Y_1 = y_1, \ldots, Y_n = y_n)$;
      $p_0 \leftarrow \Pr(X \neq 1 \mid Y_1 = y_1, \ldots, Y_n = y_n)$;
      $\mathrm{CPT}_X \leftarrow \mathrm{CPT}_X + p_1[\lambda_{X=1}] \cdot \prod_{i=1}^n [\lambda_{Y_i=y_i}] + p_0\overline{[\lambda_{X=1}]} \cdot \prod_{i=1}^n [\lambda_{Y_i=y_i}]$;
  else
    let $\operatorname{im} X = \{x_1, \ldots, x_m\}$;
    for $x \in \operatorname{im} X$ **and** $(y_1, \ldots, y_n) \in \prod_{i=1}^n \operatorname{im} Y_i$ **do**
      $p_x \leftarrow \Pr(X = x \mid Y_1 = y_1, \ldots, Y_n = y_n)$;
      $\mathrm{CPT}_X \leftarrow \mathrm{CPT}_X + p_x[\lambda_{X=x}] \cdot \prod_{i=1}^n [\lambda_{Y_i=y_i}] + \overline{[\lambda_{X=1}]} \cdot \prod_{i=1}^n [\lambda_{Y_i=y_i}]$;
    $\mathrm{CPT}_X \leftarrow \mathrm{CPT}_X \cdot (\sum_{i=1}^m [\lambda_{X=x_i}]) \cdot \prod_{i=1}^m \prod_{j=i+1}^m (\overline{[\lambda_{X=x_i}]} + \overline{[\lambda_{X=x_j}]})$;
  $\phi \leftarrow \phi \cdot \mathrm{CPT}_X$;
return ϕ;

Algorithm 1 shows how a Bayesian network with vertices $\mathcal{V}$ can be represented as a weight function $\phi\colon 2^U \to \mathbb{R}_{\geq 0}$. The algorithm begins with the unit function and multiplies it by $\mathrm{CPT}_X\colon 2^{\mathcal{E}^*(X)} \to \mathbb{R}_{\geq 0}$ for each random variable $X \in \mathcal{V}$. We call each such function a *conditional weight function* as it represents a conditional probability distribution. However, the distinction is primarily a semantic one: a function $2^{\{a,b\}} \to \mathbb{R}_{\geq 0}$ can represent $\Pr(a \mid b)$, $\Pr(b \mid a)$, or something else entirely.

For a binary random variable X, CPT_X is simply a sum of smaller functions, one for each row of the CPT. If X has more than two values, we also multiply CPT_X by some 'clause' functions that restrict the value of $\phi(T)$ to zero whenever $|\mathcal{E}(X) \cap T| \neq 1$. For the Bayesian network in Fig. 1, we get:

$$\begin{aligned}
\mathrm{CPT}_\mathrm{W} &= 0.5[\lambda_{W=1}] + 0.5\overline{[\lambda_{W=1}]} = 0.5 \cdot 1, \\
\mathrm{CPT}_\mathrm{F} &= 0.6[\lambda_{F=1}] \cdot [\lambda_{W=1}] + 0.4[\lambda_{F=0}] \cdot [\lambda_{W=1}] + 0.1[\lambda_{F=1}] \cdot [\lambda_{W=0}] + 0.9[\lambda_{F=0}] \cdot [\lambda_{W=0}] \\
&= 0.6[\lambda_{F=1}] \cdot [\lambda_{W=1}] + 0.4\overline{[\lambda_{F=1}]} \cdot [\lambda_{W=1}] + 0.1[\lambda_{F=1}] \cdot \overline{[\lambda_{W=1}]} + 0.9\overline{[\lambda_{F=1}]} \cdot \overline{[\lambda_{W=1}]}, \\
\mathrm{CPT}_\mathrm{T} &= ([\lambda_{T=l}] + [\lambda_{T=m}] + [\lambda_{T=h}]) \cdot (\overline{[\lambda_{T=l}]} + \overline{[\lambda_{T=m}]}) \cdot (\overline{[\lambda_{T=l}]} + \overline{[\lambda_{T=h}]}) \cdot (\overline{[\lambda_{T=m}]} + \overline{[\lambda_{T=h}]}) \cdot (\ldots).
\end{aligned}$$

5.1 Proof of Correctness

Introduce what this subsection is all about (and each result separately).

Lemma 2. *Let $X \in \mathcal{V}$ be a random variable with parents $\operatorname{pa}(X) = \{Y_1, \ldots, Y_n\}$. Then $\mathrm{CPT}_X\colon 2^{\mathcal{E}^*(X)} \to \mathbb{R}_{\geq 0}$ is such that for any $x \in \operatorname{im} X$ and $(y_1, \ldots, y_n) \in \prod_{i=1}^n \operatorname{im} Y_i$,*

$$\mathrm{CPT}_X(\{\lambda_{X=x}\} \cup \{\lambda_{Y_i=y_i} \mid i = 1, \ldots, n\}) = \Pr(X = x \mid Y_1 = y_1, \ldots, Y_n = y_n).$$

Proof. Let $T = \{\lambda_{X=x}\} \cup \{\lambda_{Y_i=y_i} \mid i = 1, \ldots, n\}$. If X is binary, then CPT_X is a sum of $2\prod_{i=1}^n |\operatorname{im} Y_i|$ terms, one for each possible assignment of values to variables $X, Y_1, \ldots, Y_n$. Exactly one of these terms is nonzero when applied to T, and it is equal to $\Pr(X = x \mid Y_1 = y_1, \ldots, Y_n = y_n)$ by definition.

If X is not binary, then $(\sum_{i=1}^m [\lambda_{X=x_i}])(T) = 1$, and $\left(\prod_{i=1}^m \prod_{j=i+1}^m (\overline{[\lambda_{X=x_i}]} + \overline{[\lambda_{X=x_j}]})\right)(T) = 1$, so $\mathrm{CPT}_X(T) = \Pr(X = x \mid Y_1 = y_1, \ldots, Y_n = y_n)$ by a similar argument as before. □

Proposition 1. *The function* $\phi\colon 2^U \to \mathbb{R}_{\geq 0}$ *represents the full probability distribution of the Bayesian network, i.e., if* $\mathcal{V} = \{X_1, \ldots, X_n\}$*, then*

$$\phi(T) = \begin{cases} \Pr(X_1 = x_1, \ldots, X_n = x_n) & \text{if } T = \{\lambda_{X_i = x_i} \mid i = 1, \ldots, n\} \text{ for some } (x_1, \ldots, x_n) \in \prod_{i=1}^{n} \operatorname{im} X_i \\ 0 & \text{otherwise,} \end{cases}$$

for all $T \in 2^U$*.*

Proof. If $T = \{\lambda_{X=v_X} \mid X \in \mathcal{V}\}$ for some $(v_X)_{X \in \mathcal{V}} \in \prod_{X \in \mathcal{V}} \operatorname{im} X$, then

$$\phi(T) = \prod_{X \in \mathcal{V}} \Pr\left(X = v_X \,\middle|\, \bigwedge_{Y \in \mathrm{pa}(X)} Y = v_Y\right) = \Pr\left(\bigwedge_{X \in \mathcal{V}} X = v_X\right)$$

by Lemma 2 and the definition of a Bayesian network. Otherwise there must be some non-binary random variable $X \in \mathcal{V}$ such that $|\mathcal{E}(X) \cap T| \neq 1$. If $\mathcal{E}(X) \cap T = \emptyset$, then $\left(\sum_{i=1}^{m} [\lambda_{X=x_i}]\right)(T) = 0$, and so $\mathrm{CPT}_X(T) = 0$, and $\phi(T) = 0$. If $|\mathcal{E}(X) \cap T| > 1$, then we must have two different values $x_1, x_2 \in \operatorname{im} X$ such that $\{\lambda_{X=x_1}, \lambda_{X=x_2}\} \subseteq T$ which means that $(\overline{[\lambda_{X=x_1}]} + \overline{[\lambda_{X=x_2}]})(T) = 0$, and so, again, $\mathrm{CPT}_X(T) = 0$, and $\phi(T) = 0$. □

Theorem 1. *Let* $\phi\colon 2^U \to \mathbb{R}_{\geq 0}$ *be the function generated by Algorithm 1. Then* $(\exists_U(\phi \cdot [\lambda_{X=x}]))(\emptyset) = \Pr(X = x)$*.*

Proof. Let $\mathcal{V} = \{X, Y_1, \ldots, Y_n\}$. Then

$$\begin{aligned}(\exists_U(\phi \cdot [\lambda_{X=x}]))(\emptyset) &= \sum_{T \in 2^U} (\phi \cdot [\lambda_{X=x}])(T) = \sum_{\lambda_{X=x} \in T \in 2^U} \phi(T) = \sum_{\lambda_{X=x} \in T \in 2^U} \left(\prod_{Y \in \mathcal{V}} \mathrm{CPT}_Y\right)(T) \\ &= \sum_{(y_1, \ldots, y_n) \in \prod_{i=1}^{n} \operatorname{im} Y_i} \Pr(X = x, Y_1 = y_1, \ldots, Y_n = y_n) = \Pr(X = x)\end{aligned}$$

by the following arguments:

- the proof of Theorem 1 in the ADDMC paper [10];
- if $\lambda_{X=x} \notin T \in 2^U$, then $(\phi \cdot [\lambda_{X=x}])(T) = \phi(T) \cdot [\lambda_{X=x}](T \cap \{\lambda_{X=x}\}) = \phi(T) \cdot 0 = 0$;
- Proposition 1;
- marginalisation of a probability distribution.

□

Need to make the proof nicer.

5.2 Textual Representation

Algorithm 1 encodes a Bayesian network into a function on a Boolean algebra, but how does it relate to the standard interpretation of a WMC encoding as a formula in conjunctive normal form (CNF) together with a collection of weights? The factors of ϕ that restrict the values of indicator variables for non-binary random variables are already expressed as a product of sums of 0/1-valued functions, i.e., a kind of CNF. Disregarding these functions, each conditional weight function CPT_X is represented by a sum with a term

for every subset of $\mathcal{E}^*(X)$. To encode these terms, we introduce *extended weight clauses* to the WMC format used by Cachet [17]. For instance, here is a representation of the Bayesian network from Fig. 1:

$$
\begin{array}{llll}
\lambda_{T=l} & \lambda_{T=m} & \lambda_{T=h} & 0 \\
 & -\lambda_{T=l} & -\lambda_{T=m} & 0 \\
 & -\lambda_{T=l} & -\lambda_{T=h} & 0 \\
 & -\lambda_{T=m} & -\lambda_{T=h} & 0 \\
w & \lambda_{W=1} & 0.5 & 0.5 \\
w & \lambda_{F=1} & \lambda_{W=1} & 0.6 \quad 0.4 \\
w & \lambda_{F=1} & -\lambda_{W=1} & 0.1 \quad 0.9 \\
w & \lambda_{T=l} & \lambda_{W=1} & 0.2 \quad 1 \\
w & \lambda_{T=m} & \lambda_{W=1} & 0.4 \quad 1 \\
w & \lambda_{T=h} & \lambda_{W=1} & 0.4 \quad 1 \\
w & \lambda_{T=l} & -\lambda_{W=1} & 0.6 \quad 1 \\
w & \lambda_{T=m} & -\lambda_{W=1} & 0.3 \quad 1 \\
w & \lambda_{T=h} & -\lambda_{W=1} & 0.1 \quad 1
\end{array}
$$

where each indicator variable is eventually replaced with a unique positive integer. Each line prefixed with a w can be split into four parts: the 'main' variable (always not negated), conditions (possibly none), and two weights. For example, the line

$$w \quad \lambda_{T=m} \quad -\lambda_{W=1} \quad 0.3 \quad 1$$

encodes the function $0.3[\lambda_{T=m}] \cdot \overline{[\lambda_{W=1}]} + 1\overline{[\lambda_{T=m}]} \cdot \overline{[\lambda_{W=1}]}$ and can be interpreted as defining two conditional weights: $\nu(T = m \mid W = 0) = 0.3$, and $\nu(T \neq m \mid W = 0) = 1$, the former of which corresponds to a row in the CPT of T while the latter is artificially added as part of the encoding. In our encoding of Bayesian networks, it is always the case that, in each weight clause, either both weights sum to one, or the second weight is equal to one. Finally, note that (without any additional restrictions), the measure induced by these weight functions is not probabilistic (i.e., $\mu(\top)$ may not be equal to one).

5.3 Changes to ADDMC

ADDMC constructs the *Gaifman graph* [12] of the input CNF formula as an aid for the algorithm's heuristics. This graph has as vertices the variables of the formula, and there is an edge between two variables u and v if there is a clause in the formula that contains both u and v. We extend this definition to functions on Boolean algebras, i.e., the factors of ϕ. For any pair of distinct variables $u, v \in U$, we draw an edge between them in the Gaifman graph if there is a function $\alpha \colon 2^X \to \mathbb{R}_{\geq 0}$ that is a factor of ϕ such that $u \in X$ and $v \in X$. For instance, a factor such as CPT_X will enable edges between all distinct pairs of variables in $\mathcal{E}^*(X)$.

Even though the function ϕ produced by Algorithm 1 is constructed to have 2^U as its domain, sometimes the domain is effectively reduced to 2^V for some $V \subset U$ by the ADD manipulation algorithms that optimise the ADD representation of a function. For a simple example, consider $\alpha : 2^{\{a\}} \to \mathbb{R}_{\geq 0}$ defined as $\alpha(\{a\}) = \alpha(\emptyset) = 0.5$. Then α can be reduced to $\alpha' \colon 2^{\emptyset} \to \mathbb{R}_{\geq 0}$ defined as $\alpha'(\emptyset) = 0.5$. To compensate for these reductions, for the original WMC format with a weight function $w \colon U \cup \{\neg u \mid u \in U\} \to \mathbb{R}_{\geq 0}$, ADDMC would multiply its computed answer by $\prod_{u \in U \setminus V} w(u) + w(\neg u)$. With the new WMC format, we instead multiply the answer by $2^{|U \setminus V|}$. Each 'excluded' variable $u \in U \setminus V$ satisfies two properties: all weights associated with u are equal to 0.5 (otherwise the corresponding CPT would depend on u, and u would not be excluded), and all other CPTs are independent of u (or they may have a trivial dependence, where the probability stays the same if u is replaced with its complement). Thus, the CPT that corresponds to u still multiplies every model by 0.5, but the number of models being considered by the ADDMC is halved. To correct for this, we multiply the final answer by two for every $u \in U \setminus V$.

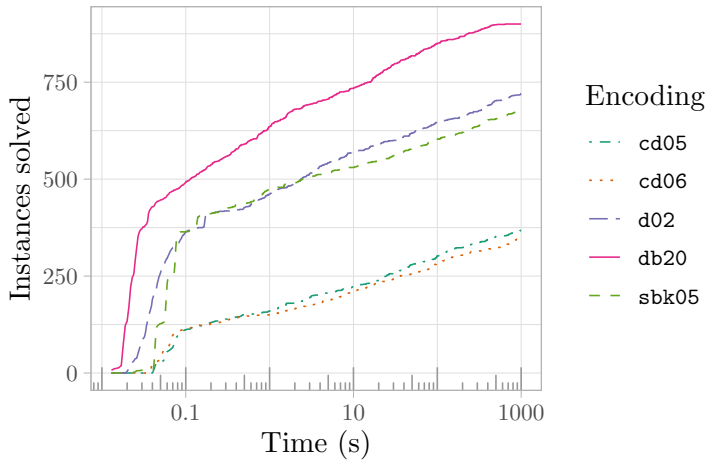


Figure 2: Cumulative number of instances solved by ADDMC over time using each encoding

Table 3: The numbers of instances solved by ADDMC with each encoding (uniquely, faster than with others, and in total) under the described time and memory constraints (out of 1216 instances)

Encoding	Unique	Fastest	Total
cd05	2	3	372
cd06	0	1	351
d02	41	99	726
db20	228	871	901
sbk05	6	36	687

6 Experimental Comparison

- We chose not to measure or compare encoding time because of different languages of implementation and the fact that our encoding algorithm (i.e., the process that converts a Bayesian network into a textual encoding such as in Section 5.2) is linear in the total number of CPT rows and so is unlikely to be slower than, e.g., cd06, which relies on solving NP-complete problems as part of the encoding process [4].
- Whenever a Bayesian network comes with an evidence file, we compute the probability of evidence. Otherwise, let X denote the last-mentioned vertex in the Bayesian network. If true is a valid value of X, we compute the marginal probability of $X =$ true. Otherwise, we pick the value of X which is listed first and calculate its marginal probability.
- The experiments were run on Intel Xeon Gold 6138 processor with an 8 GB memory limit.
- For all other encodings, we use their implementation in Ace 3.0[2] with `-encodeOnly` and `-noEclause` flags. However, Ace was not used to encode evidence, as preliminary experiments revealed that the evidence-encoding implementation contains bugs that can lead to incorrect answers or a Java exception being thrown on some instances of the data set (and the source code is not publicly available). Instead, we simply list all the evidence as additional clauses in the encoding (regardless of which encoding is used).
- Note that cd05 and cd06 purposefully produce overly relaxed encodings that contain extra models and thus yield incorrect probabilities [3, 4]. These additional models are supposed to be filtered out during circuit compilation [3], but this is not easily achievable with ADDMC. Nonetheless, we include both encodings in our timing experiments.

Data. For experiments, we use the Bayesian networks available with Ace and Cachet[3], most of which happen to be binary. We classify them into the following seven categories: • DQMR and • Grid networks as described by Sang et al. [18], • Friends and Smokers, • Mastermind, and • Random Blocks from the work of Chavira et al. [7], • remaining binary Bayesian networks that include Plan Recognition [18], Students

[2] http://reasoning.cs.ucla.edu/ace/
[3] https://www.cs.rochester.edu/u/kautz/Cachet/

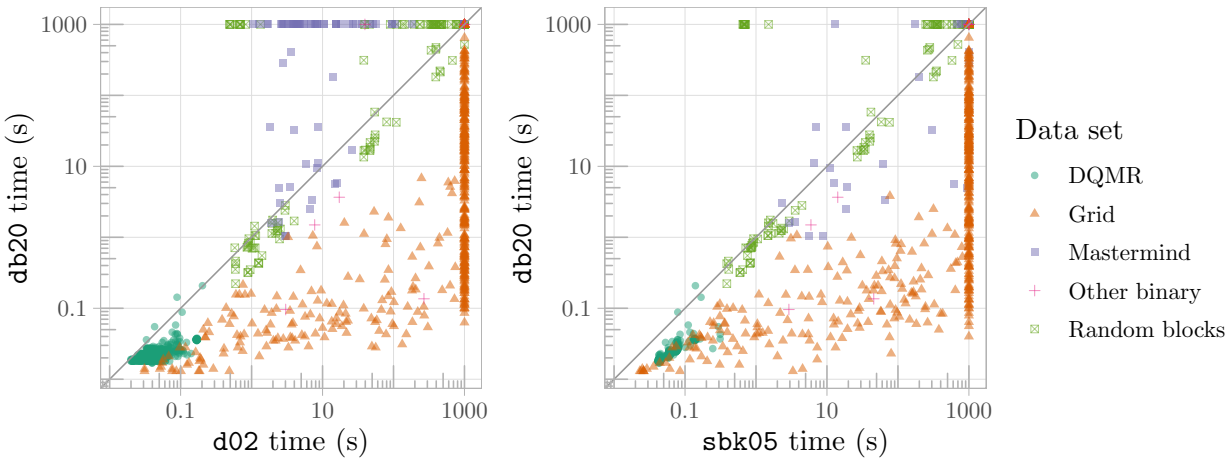


Figure 3: ADDMC inference time using `db20` compared to `d02` (left) and `sbk05` (right) on an instance-by-instance basis across all data sets.

Table 4: Asymptotic upper bounds on the numbers of variables and clauses/ADDs for each encoding

Encoding(s)	Variables	Clauses/ADDs
`cd05`, `cd06`, `sbk05`	$\mathcal{O}(nv^{d+1})$	$\mathcal{O}(nv^{d+1})$
`d02`	$\mathcal{O}(nv^{d+1})$	$\mathcal{O}(ndv^{d+1})$
`db20`	$\mathcal{O}(nv)$	$\mathcal{O}(nv^2)$

and Professors [7], and `tcc4f`, and • non-binary classic Bayesian networks (`alarm`, `diabetes`, `hailfinder`, `mildew`, `munin1`–4, `pathfinder`, `pigs`, `water`).

Observations.

- The order of the encodings from best to worst is exactly the opposite of that in the previous literature.
- Our encoding is particularly promising on instances of Grid networks. They are set up so that the entire Bayesian network is relevant to the query.
- However, `db20` struggles with instances from Mastermind and Random Blocks domains (but so does `sbk05`). We conjecture that this is so either because the particular structure of these problems confuse the heuristics used by ADDMC or because these instances allow for significant simplifications that are exploited by `d02` but not `db20`.
- An observation from the cumulative plot: `db20` solves the same number of instances in 6.374 s as `d02` does in 1000 s.

It could be confusing to use the word 'variable' here. I'm using it as a synonym to 'logical atom'.

Explaining the performance benefits. Let $n = |\mathcal{V}|$ be the number of vertices in the Bayesian network, $d = \max_{X \in \mathcal{V}} |\mathrm{pa}(X)|$ the maximum in-degree (i.e., number of parents), and $v = \max_{X \in \mathcal{V}} |\operatorname{im} X|$ the maximum number of values per variable. Table 4 shows how `db20` has both fewer variables and fewer ADDs than any other encoding. However, note that these are upper bounds and most encodings (including `db20`) can be smaller in certain situations (e.g., with binary random variables or when a CPT has repeating probabilities).

We equate clauses and ADDs (more specifically, factors of the function ϕ from Algorithm 1) here because ADDMC interprets each clause of any WMC encoding as a multiplicative factor of the ADD that represents the entire WMC instance [10]. For literal-weight encodings, each weight is also a factor, but that has no effect on the asymptotic bounds in the table.

7 Conclusions and Future Work

- Bayesian networks and ADDMC are only particular examples. This should also work with Cachet [17].
  - Potential criticism may be that this doesn't allow us to use SAT-based techniques for probabilistic inference. However, they can still be used for a significant part of the encoding.
    - Zero-probability weights and one-probability weights can be interpreted as logical clauses. This doesn't affect ADDMC but could be useful for other solvers.
- Extra benefit: one does not need to come up with a way to turn some probability distribution to into a fully independent one.
- Important future work: replacing ADDs with AADDs [19] is likely to bring performance benefits. Other extensions:
  - FOADDs can represent first order statements;
  - XADDs can replace WMI for continuous variables;
  - ADDs with intervals can do approximations.
- Filtering out ADDs that have nothing to do with the answer helps tremendously, but I'm purposefully not doing that.
- Other things can be compiled into WMC, e.g., probabilistic programs [14], ProbLog [11].

Acknowledgements. This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF) (`http://www.ecdf.ed.ac.uk/`).

References

[1] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. *Formal Methods Syst. Des.*, 10(2/3):171–206, 1997.

[2] Craig Boutilier, Nir Friedman, Moisés Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In Eric Horvitz and Finn Verner Jensen, editors, *UAI '96: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence, Reed College, Portland, Oregon, USA, August 1-4, 1996*, pages 115–123. Morgan Kaufmann, 1996.

[3] Mark Chavira and Adnan Darwiche. Compiling Bayesian networks with local structure. In Kaelbling and Saffiotti [15], pages 1306–1312.

[4] Mark Chavira and Adnan Darwiche. Encoding CNFs to empower component analysis. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 2006.

[5] Mark Chavira and Adnan Darwiche. Compiling Bayesian networks using variable elimination. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2443–2449, 2007.

[6] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

[7] Mark Chavira, Adnan Darwiche, and Manfred Jaeger. Compiling relational Bayesian networks for exact inference. *Int. J. Approx. Reason.*, 42(1-2):4–20, 2006.

[8] Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *J. Appl. Non Class. Logics*, 11(1-2):11–34, 2001.

[9] Adnan Darwiche. A logical approach to factoring belief networks. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness, and Mary-Anne Williams, editors, *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, pages 409–420. Morgan Kaufmann, 2002.

[10] Jeffrey M. Dudek, Vu Phan, and Moshe Y. Vardi. ADDMC: weighted model counting with algebraic decision diagrams. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1468–1476. AAAI Press, 2020.

[11] Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. Inference in probabilistic logic programs using weighted CNF's. In Fábio Gagliardi Cozman and Avi Pfeffer, editors, *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 211–220. AUAI Press, 2011.

[12] Haim Gaifman. On local and non-local properties. In *Studies in Logic and the Foundations of Mathematics*, volume 107, pages 105–135. Elsevier, 1982.

[13] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. SPUDD: stochastic planning using decision diagrams. In Kathryn B. Laskey and Henri Prade, editors, *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999*, pages 279–288. Morgan Kaufmann, 1999.

[14] Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. Dice: Compiling discrete probabilistic programs for scalable inference. *CoRR*, abs/2005.09089, 2020.

[15] Leslie Pack Kaelbling and Alessandro Saffiotti, editors. *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Professional Book Center, 2005.

[16] Thomas D. Nielsen, Pierre-Henri Wuillemin, Finn Verner Jensen, and Uffe Kjærulff. Using ROBDDs for inference in Bayesian networks with troubleshooting as an example. In Craig Boutilier and Moisés Goldszmidt, editors, *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000*, pages 426–435. Morgan Kaufmann, 2000.

[17] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004.

[18] Tian Sang, Paul Beame, and Henry A. Kautz. Performing Bayesian inference by weighted model counting. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 475–482. AAAI Press / The MIT Press, 2005.

[19] Scott Sanner and David A. McAllester. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In Kaelbling and Saffiotti [15], pages 1384–1390.

[20] Fabio Somenzi. CUDD: CU decision diagram package release 3.0.0. *University of Colorado at Boulder*, 2015.

[21] Han Zhao, Mazen Melibari, and Pascal Poupart. On the relationship between sum-product networks and Bayesian networks. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 116–124. JMLR.org, 2015.